CS1201: Data Structures and Algorithms

Optimal Freelancer-Project Matching System

Technical Documentation and Implementation Report

Submitted By:

Name: Shreyas Chhabra

Roll Number: 2401MC01

Group: 19

Submitted To:

Prof. Jimson Mathew
Department of Computer Science

Indian Institute of Technology, Patna

Contents

1 Executive Summary

This report presents a comprehensive analysis of the Optimal Freelancer-Project Matching System, an advanced solution designed to efficiently match freelancers with suitable projects using sophisticated algorithms and data structures. The system implements the Hungarian Algorithm for optimal matching and utilizes Bloom filters for efficient skill verification.

2 List of Data Structures and Algorithms Used

2.1 Data Structures

1. **Bloom Filter**
   - Purpose: Efficient skill lookup
   - Implementation: Bit array with multiple hash functions
   - Space Complexity: O(m), where m is the filter size
2. **Bipartite Graph**
   - Purpose: Represent freelancer-project relationships
   - Implementation: Adjacency list
   - Space Complexity: O(V + E)

2.1.1 Bipartite Graph

```c
typedef struct GraphNode {
    int id;
    int weight;
    struct GraphNode* next;
} GraphNode;

typedef struct {
    GraphNode** adjacency_list;
    int num_nodes;
    int* node_types;
} BipartiteGraph;
```

Listing 1: Bipartite Graph Structure

3. **Cost Matrix**
   - Purpose: Store matching weights for Hungarian Algorithm
   - Implementation: 2D dynamic array
   - Space Complexity: $O(n^2)$

2.2 Algorithms

1. **Hungarian Algorithm**
   - Purpose: Optimal assignment
   - Time Complexity: $O(n^3)$
   - Modifications: Adapted for skill matching
2. **Hash Functions**

- DJB2, SDBM, and Lose Lose hash
- Purpose: Bloom filter implementation
- Time Complexity: O(k) per operation

3 Project Setup and Execution

3.1 Prerequisites

- GCC Compiler (Version 7.0+)
- Make build system
- Modern web browser
- Python 3.7+ (for data generation scripts)

3.2 Building the Project

```
# Clone the repository
git clone <repository-url>
cd freelancer-matching

# Build the backend
cd backend
make clean
make all

# Start the server
./freelancer_matcher
```

3.3 Running the Frontend

- Open frontend/index.html in a web browser
- Server should be running on localhost:8080
- Data files should be present in the data/ directory

4 Hungarian Algorithm: Working Example

Let's trace through a small example with 3 freelancers and 3 projects:

4.1 Initial Cost Matrix

	P1	P2	P3
F1	40	60	75
F2	25	35	55
F3	55	30	45

Table 1: Initial cost matrix based on skill mismatch

4.2 Step 1: Row Reduction

	P1	P2	P3
F1	0	20	35
F2	0	10	30
F3	25	0	15

Table 2: After subtracting row minimums

4.3 Step 2: Column Reduction

	P1	P2	P3
F1	0	20	20
F2	0	10	15
F3	25	0	0

Table 3: After subtracting column minimums

4.4 Final Assignment

- F1 → P1 (Cost: 40)
- F2 → P2 (Cost: 35)
- F3 → P3 (Cost: 45)

5 Bloom Filter: Working Example

5.1 Test Case

Consider a Bloom filter for skill verification:

```
// Initialize Bloom filter (size: 1024 bits)
BloomFilter filter;
bloom_init(&filter);

// Add skills
bloom_add(&filter, "Python");
bloom_add(&filter, "JavaScript");
bloom_add(&filter, "React");

// Test membership
printf("Has Python? %d\n", bloom_check(&filter, "Python"));      // 1
printf("Has Java? %d\n", bloom_check(&filter, "Java"));          // 0
printf("Has JavaScript? %d\n", bloom_check(&filter, "JavaScript")); // 1
```

5.2 Hash Values Example

For skill "Python":

- DJB2 Hash: 5381 → 193485876 mod 1024 = 724
- SDBM Hash: 0 → 193487098 mod 1024 = 138
- Lose Lose Hash: 0 → 612 mod 1024 = 612

6 System Architecture

This report presents a comprehensive analysis of the Optimal Freelancer-Project Matching System, an advanced solution designed to efficiently match freelancers with suitable projects using sophisticated algorithms and data structures. The system implements the Hungarian Algorithm for optimal matching and utilizes Bloom filters for efficient skill verification.

6.1 Project Structure

Directory Structure

```
freelancer-matching/
 backend/
    bloom_filter.c
    match_allocator.c
    utils.c
    main.c
 frontend/
    index.html
    script.js
    styles.css
 data/
     freelancers.csv
     projects.csv
     availability.csv
```

6.2 Backend Components

The backend is implemented in C, providing high performance for computational tasks. Key components include:

- **HTTP Server**: Implemented using POSIX sockets (Port 8080)
- **Matching Engine**: Hungarian Algorithm implementation
- **Bloom Filter Service**: Efficient skill verification
- **Data Processor**: CSV file handling and data structures

6.3 Frontend Components

The frontend provides an interactive dashboard implemented using:

- HTML5 for structure
- CSS3 for responsive design
- JavaScript (ES6+) for dynamic interactions
- Chart.js for data visualization

7 Implementation Details

7.1 Hash Functions

The system uses three hash functions for the Bloom filter:

```
// DJB2 Hash
hash = ((hash << 5) + hash) + c

// SDBM Hash
hash = c + (hash << 6) + (hash << 16) - hash

// Lose Lose Hash
hash += c
```

Listing 2: Hash Functions

7.2 Frontend-Backend Integration

API Integration

```
GET /matches
Response: {
    "total_freelancers": n,
    "total_projects": m,
    "matches": [{
        "freelancer": {...},
        "project": {...},
        "score": x
    }]
}
```

7.3 Matching Process

1. Load freelancer and project data
2. Generate compatibility scores
3. Apply Hungarian Algorithm
4. Return optimal matches

8 Performance Analysis

8.1 Time Complexity

- Hungarian Algorithm: $O(n^3)$
- Bloom Filter Operations: O(1)
- Graph Operations: O(E) where E is number of edges

8.2 Space Complexity

- Bloom Filter: O(m) where m is filter size
- Bipartite Graph: O(V + E)
- Cost Matrix: $O(n^2)$

9 Future Enhancements

9.1 Algorithm Improvements

- Implementation of machine learning for skill matching
- Enhanced scoring algorithm with historical data
- Real-time matching updates

9.2 Feature Additions

- Skill recommendation system
- Project timeline optimization
- Team formation suggestions

9.3 Technical Improvements

- Distributed computing support
- Cache implementation
- WebSocket integration for real-time updates

10 Conclusion

The Optimal Freelancer-Project Matching System demonstrates the effective use of advanced algorithms and data structures to solve a complex resource allocation problem. The combination of the Hungarian Algorithm, Bloom filters, and bipartite graph representation provides an efficient and scalable solution.